1. How to debug pod in Kubernetes

Ans: kubectl get pods

kubectl describe pod <pod-name>

2.How to delete pod

1. **Delete a Pod by Name**:

kubectl delete pod <pod-name>

2. **Delete Pods Using Labels**:

kubectl delete pods -l <label-key>=<label-value>

3. **Delete Pods in a Namespace**

**kubectl delete pod -n <namespace> <pod-name>**

4. Delete All Pods in a Namespace

kubectl delete pods --all -n <namespace>

3. **How to restore pods in k8s?**

In Kubernetes, restoring pods typically involves recreating or scaling the deployment, statefulset, or replication controller that manages the pods. The exact steps may vary depending on the type of workload you are using. Here are general steps for different types of workloads:

**Deployments:**

If you're using Deployments, you can update the deployment to trigger the creation of new pods. The old pods will be terminated, and new ones will be created with the updated configuration.

**StatefulSets:**

For StatefulSets, you can also update the StatefulSet to trigger pod recreation. However, Stateful Sets provide stable network identities for pods, so rolling updates may be more gradual.

**Replication Controllers:**

If you are using Replication Controllers (although Deployments are more commonly used), you can scale the replication controller to a desired number of replicas

kubectl scale rc your-replication-controller --replicas=3

5. **How to execute or build docker image**

1.Create docker file

# Use an official Node.js runtime as a base image

FROM node:14

# Set the working directory in the container

WORKDIR /usr/src/app

# Copy package.json and package-lock.json to the working directory

COPY package\*.json ./

# Install app dependencies

RUN npm install

# Copy the application files to the working directory

COPY . .

# Expose a port for the application

EXPOSE 3000

# Define the command to run the application

CMD ["npm", "start"]

2.Build the docker image

docker build -t your-image-name:tag .

3. **Verify the Built Image:**

docker images

4. **Run a Container Based on the Image:**

docker run -p 8080:3000 your-image-name:tag

What is docker **prune**

1.Remove All Unused Resources:

docker system prune

2. **Remove Only Dangling Images:**

docker image prune

3. **Remove Volumes Not Used by at Least One Container:**

docker volume prune

**4.Remove Networks Not Used by at Least One Container:**

docker network prune

**6.**Key Differences difference b/w git fetch and git pull?

* **git fetch** only downloads changes from the remote repository but does not automatically integrate them into your local branch.
* **git pull** is a combination of **git fetch** followed by an automatic merge, potentially leading to merge conflicts.
* **git fetch** is typically used when you want to review changes before integrating them or when you want to update your local repository without affecting your current branch.
* **git pull** is used when you want to quickly update your local branch with the latest changes from the remote.

7. What are azure Artifacts?

Azure Artifacts enables developers to share their code efficiently and manage all their packages from one place. With Azure Artifacts, developers can publish packages to their feeds and share it within the same team, across organizations, and even publicly.

1. **Centralized Package Management**:
   * Azure Artifacts provides a centralized location to store and manage all your organization's software artifacts and packages.
2. **Support for Multiple Package Types**:
   * It supports various package formats, including NuGet, npm, Maven, and Python packages. This allows you to manage dependencies for a wide range of projects and applications.
3. **Private Feeds for Secure Package Storage**:
   * You can create private feeds to store sensitive or proprietary packages securely within your organization. This ensures that only authorized team members can access and use these packages.

8.What is Different b/w Variable and Variable Group?

Variables:

Scope: Variables are defined at the pipeline level. They are specific to a single pipeline and can be used within that pipeline and its stages, jobs, and tasks.

Syntax :

variables:

myVariable: 'someValue'

Variable Groups:

Scope: Variable groups are defined at the project or organization level. They can be used across multiple pipelines within the same project or across projects in the organization

Syntax:

variables:

- group: 'MyVariableGroup'

AKS

9.What is different b/w **Deployments**, **StatefulSets** and **Daemonsets**?

**Deployments** are great for stateless applications that can be easily scaled horizontally**,**

**StatefulSets** are great for applications that require persistent storage and have state that needs to be maintained.

**DaemonSets** are great for running an application on every node in the cluster**.**

**Deployments:**

* **Purpose:** Deployments are a high-level abstraction that manages ReplicaSets. They are primarily used for stateless applications.
* **Scaling:** Deployments support easy scaling and rolling updates. They ensure that a specified number of replicas of the application are always running.
* **Pods:** Pods managed by Deployments are typically interchangeable, and the application is designed to be stateless.

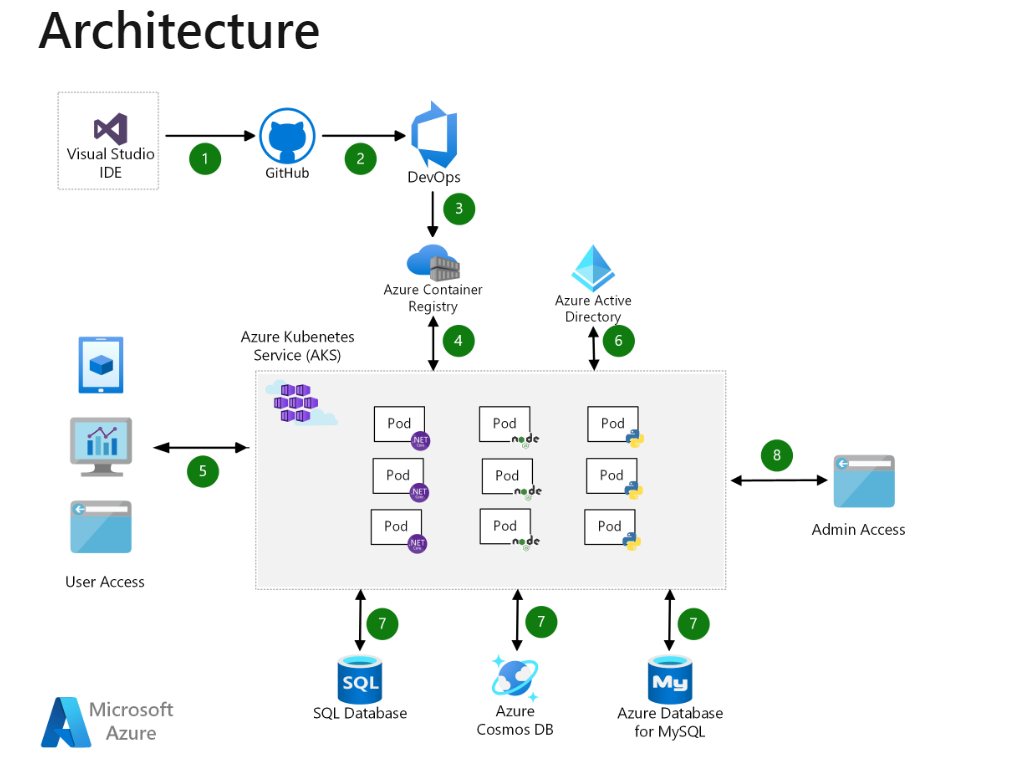
**StatefulSets:**

* **Purpose:** StatefulSets are designed for stateful applications that require stable network identifiers and stable storage. Examples include databases.
* **Pods:** StatefulSets assign a unique and persistent hostname to each pod, and they manage the order of pod creation and deletion.
* **Scaling:** Scaling with StatefulSets is typically done manually, and they are suitable for applications that require a fixed network identity.

**DaemonSets:**

* **Purpose:** DaemonSets ensure that a copy of a pod runs on each node in the cluster. They are typically used for cluster-level services like log collectors or monitoring agents.
* **Pods:** Each node runs exactly one copy of the pod managed by the DaemonSet.
* **Scaling:** DaemonSets automatically scale as nodes are added or removed from the cluster.





What role we have access in azure?

**Common RBAC Role:**

1. **Owner:**
   * Has full access to all resources, including the right to delegate access to others.
2. **Contributor:**
   * Can create and manage all types of Azure resources but can't grant access to others.
3. **Reader:**
   * Can view resources but cannot make any changes.
4. **User Access Administrator:**
   * Grants the ability to manage user access to Azure resources.
5. **Security Administrator:**
   * Manages identity and access management components like Azure AD roles.

What is Azure Active Directory ?

Azure Active Directory (Azure AD) is Microsoft's cloud-based identity and access management service. It is designed to help your organization securely manage and authenticate users for access to various resources, applications, and services both within the Microsoft 365 ecosystem and external applications.

Key features and aspects of Azure Active Directory include:

1. **Identity Management:**
   * Azure AD provides identity services that enable you to create and manage user accounts and groups. This includes features such as user provisioning, self-service password reset, and group-based access control.
2. **Single Sign-On (SSO):**
   * Azure AD supports Single Sign-On, allowing users to sign in once and access multiple applications without the need to re-enter credentials. This simplifies the user experience and enhances security.
3. **Multi-Factor Authentication (MFA):**
   * To enhance security, Azure AD supports multi-factor authentication. This requires users to provide additional verification beyond their password, such as a phone code or biometric verification.

10. Explain Kubernetes architecture?

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. The architecture of Kubernetes is designed to be scalable, extensible, and resilient. Here's an overview of the key components in the Kubernetes architecture.

1. **Master Node:**
   * **API Server:** The API server is the central management entity that exposes the Kubernetes API. It processes RESTful API requests, validates them, executes corresponding operations, and updates the etcd datastore.
   * **Controller Manager:** The Controller Manager is responsible for regulating the state of the system. It includes various controllers that watch for changes in the cluster state (e.g., nodes, pods) and take corrective action to ensure the desired state is maintained.
   * **Scheduler:** The Scheduler is responsible for placing containers onto available nodes based on resource requirements, policies, and constraints. It makes decisions on where to deploy new pods.
   * **etcd Datastore:** etcd is a distributed key-value store that acts as the cluster's source of truth for all configuration data. It stores the desired state of the cluster and serves as the persistent storage backend for all cluster data.
2. **Worker Nodes:**
   * **Kubelet:** The Kubelet is an agent that runs on each node in the cluster. It is responsible for communicating with the API server, managing the container lifecycle (starting, stopping, restarting), and ensuring that containers are healthy.
   * **Kube Proxy:** Kube Proxy maintains network rules on nodes, allowing communication between different pods and external network entities. It performs network address translation (NAT) and load balancing for services.
3. **Pods:**
   * **Pods:** Pods are the smallest deployable units in Kubernetes. They are logical groups of one or more containers that share the same network namespace and storage. Containers within a pod communicate over the localhost, making them suitable for closely coupled processes.
4. **Service:**
   * **Service:** A Kubernetes Service is an abstraction that exposes a set of pods as a network service. It provides a stable IP address and DNS name, enabling other applications to discover and communicate with the pods.
5. **Volume:**
   * **Volume:** Volumes provide persistent storage to containers within a pod. They can be used to share data between containers, persist data beyond the lifecycle of a pod, or provide storage for stateful applications.
6. **Namespace:**
   * **Namespace:** Namespaces provide a way to create virtual clusters within a physical cluster. They help in organizing and isolating resources, allowing multiple users or teams to use the same cluster without interfering with each other.

Documentation link: <https://kubernetes.io/docs/concepts/overview/components/>

**11.What Types of Services in K8s?**

* **ClusterIP:** Exposes the Service on a cluster-internal IP address. This type is accessible only within the cluster.
* **NodePort:** Exposes the Service on each Node's IP address at a static port. This type allows external access to the Service.
* **LoadBalancer:** Creates an external load balancer in cloud environments (e.g., AWS, GCP, Azure) and assigns a stable external IP address to the Service.
* **ExternalName:** Maps the Service to the contents of the **externalName** field (e.g., DNS name).

1. What is Default Service?

Cluster IP

b. What is different NodePort and Load balancer?

**Key Differences:**

* **NodePort:**
  + Exposes the service on each node at a static port.
  + Direct access using any node's IP address and assigned NodePort.
  + Suitable for development and testing environments.
* **LoadBalancer:**
  + Uses the cloud provider's load balancer to expose the service.
  + Cloud provider assigns an external IP address that routes traffic to the service.
  + Suitable for production environments requiring a scalable and highly available external endpoint.

1.What is replication controller and ReplicaSet?

Ans: **Replication Controller Vs Replica Set:**

| **Replication Controller** | **Replica Set** |
| --- | --- |
| The Replication Controller is the original form of replication in Kubernetes | ReplicaSets are a higher-level API that gives the ability to easily run multiple instances of a given pod |
| The Replication Controller uses **equality-based selectors** to manage the pods. | ReplicaSets Controller uses **set-based selectors** to manage the pods. |
| The rolling-update command works with Replication Controllers | The rolling-update command won’t work with ReplicaSets. |
| Replica Controller is deprecated and replaced by ReplicaSets. | Deployments are recommended over ReplicaSets. |

12.What is difference b/w Container and Pod?

**Container:**

* **Definition:** A container is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools.
* **Characteristics:**
  + Containers provide process isolation, ensuring that applications run consistently across different environments.
  + They encapsulate an application and its dependencies, allowing it to run reliably and consistently across different environments.
  + Common container runtimes include Docker, containerd, and others.

**Pod:**

* **Definition:** A pod is the smallest deployable unit in Kubernetes and represents a single instance of a running process in a cluster.
* **Characteristics:**
  + A pod can contain one or more containers that share the same network namespace, storage, and have an IP address in the cluster.
  + Containers within the same pod can communicate with each other using localhost, making them co-located and tightly coupled.
  + Pods provide a higher level of abstraction than individual containers and are used to deploy and manage containers together.

13. What is difference b/w Virtualization and Containerization?

* **Virtualization** creates virtual machines with their own operating systems, providing strong isolation and pre-allocated resources.
* **Containerization** uses lightweight containers that share the host OS kernel, offering efficient resource utilization and faster deployment times.

14.What is Copy and Add Command?

**Basic Functionality:**

* **COPY** is a straightforward command used to copy files and directories from the source on the host to the destination in the container.

ADD:

Extended Functionality:

ADD has additional functionality compared to COPY. It can also be used to:

Extract tarballs during the copy.

Copy files from a URL.

ADD ./archive.tar.gz /usr/src/

ADD [http://example.com/file.txt /usr/src/](http://example.com/file.txt%20/usr/src/)

1.What is terraform state file and why is it important

Terraform logs information about the resources it has created in a state file. This enables Terraform to know which resources are under its control and when to update and destroy them. The terraform state file, by default, is named terraform.tfstate and is held in the same directory where Terraform is run. It is created after running terraform apply.

2.What is workspace in terraform ?

3. What does the terraform init do?

1. **Backend Initialization:**
   * It initializes the backend, which is the location where Terraform stores its state files. The backend can be local (default), or it can be set to a remote system like AWS S3, Azure Storage, or Terraform Cloud.
2. **Plugin Download:**
   * It downloads the necessary provider plugins specified in the configuration. Providers are responsible for interacting with the APIs of the target infrastructure, such as AWS, Azure, or Google Cloud.
3. **Module Installation:**
   * If the configuration uses modules, it downloads and installs the modules specified in the configuration. Modules are reusable components that can be used to encapsulate and organize Terraform configurations.
4. **Initialization of the Working Directory:**
   * It sets up the working directory for Terraform, creating the necessary hidden directories and files.
5. **Verification:**
   * It verifies the syntax and validity of the configuration files in the working directory.
6. **Remote State Configuration:**
   * If using remote state, it configures the backend with the necessary settings, such as the backend type, access credentials, and other configuration parameters.
7. What is the difference between plan and apply?

**terraform plan:**

* Analysis the current state of the infrastructure and compares it with the desired state specified in your Terraform configuration files. Outputs a detailed report showing what will be added, modified, or deleted.
* No changes are made to the actual infrastructure during this step.
* It's a preview or dry run of the proposed modifications.

Terraform apply:

* **Action:** Enacts the changes by creating, updating, or deleting resources as needed to align the infrastructure with the desired state.
* **Changes:** Actual modifications are made to the infrastructure during this step based on the approved execution plan.

Additional Info:

**Backup State Files**: Before applying significant changes, especially in production environments, consider backing up your Terraform state files to facilitate rollback if needed.

**Interactive Confirmation**: terraform apply is interactive by default and requires user confirmation before proceeding. Review the output and confirm with "yes" if the changes are as expected.

**Non-Interactive Mode (Auto-Approval):** In automated environments or CI/CD pipelines, use the -auto-approve flag to skip the interactive confirmation step:

‘’’ terraform apply -auto-approve’’’

3.How do we secure terraform state file?

**Use Remote Backends:**

* Store the Terraform state remotely using a secure backend. Options include cloud storage services like AWS S3, Azure Storage, or Terraform Cloud. Remote backends often provide features like encryption and access controls.

**Use State Locking**:

* Employ state locking mechanisms to prevent concurrent modifications. This is especially important in team environments or when multiple instances of Terraform are running simultaneously.
* Remote backends like Terraform Cloud and Azure Storage support state locking.

**State locking in Terraform** is a mechanism designed to prevent concurrent access and modifications to the Terraform state file by multiple users or processes.

When multiple instances of Terraform attempt to apply changes to the same infrastructure simultaneously, state locking ensures that only one instance can modify the state file at a time. This prevents conflicts and data corruption that could occur if changes were made concurrently.

1. **Concurrency Control:**
   * State locking is essential in scenarios where multiple team members or automation processes may be working with the same infrastructure. It prevents the "lost update" problem that could occur when multiple Terraform instances try to modify the state file simultaneously.
2. **Locking Mechanisms:**
   * Terraform supports different backend types for storing the state, and the availability of state locking depends on the chosen backend. For example:
     + **Remote Backends (e.g., AWS S3, Azure Storage, Terraform Cloud):** These often support built-in state locking mechanisms.
     + **Local Backend:** Local backends do not support built-in locking, and users are responsible for managing it externally if needed.
3. **Lock and Unlock Operations:**
   * When a Terraform operation (such as **terraform apply**) starts, it attempts to acquire a lock on the state file. If successful, it proceeds with the operation. Once the operation is complete, the lock is released.
   * If a lock cannot be acquired (because another Terraform process holds the lock), the operation waits until the lock is released or times out.
4. **Locking Timeouts:**
   * Locks have a timeout period, and if a lock cannot be acquired within the specified timeout, Terraform considers it a failure. This prevents situations where a long-running process holds a lock indefinitely.
5. **Backend Configuration:**
   * Locking configuration might be specific to the backend in use. For example, with AWS S3, you can enable versioning to track changes and use DynamoDB tables for locking.

5.How do you define local variables in terraform?

**Local variables can be declared once and used any number of times in the module**. These can be accessed as objects by using the format of “local.Variable\_Name”.

**Unlike variables found in programming languages, Terraform’s locals don’t change values during or between Terraform runs such as plan, apply, or destroy.**

Local Variables:

A local value assigns a name to an [expression](https://developer.hashicorp.com/terraform/language/expressions), so you can use the name multiple times within a module instead of repeating the expression.

If you're familiar with traditional programming languages, it can be useful to compare Terraform modules to function definitions:

* [Input variables](https://developer.hashicorp.com/terraform/language/values/variables) are like function arguments.
* [Output values](https://developer.hashicorp.com/terraform/language/values/outputs) are like function return values.
* Local values are like a function's temporary local variables.

**locals {**

**service\_name = "forum"**

**owner = "Community Team "**

**}**

**locals {**

**# Ids for multiple sets of EC2 instances, merged**

**instance\_ids = concat(aws\_instance.blue.\*.id, aws\_instance.green.\*.id)**

**}**

**locals {**

**# Common tags to be assigned to all resources**

**common\_tags = {**

**Service = local.service\_name**

**Owner = local.owner**

**}**

**}**

**What is data block**

Azure Cloud:

1. **Service Endpoint:**
   * **Definition:** A Service Endpoint is a configuration that **allows virtual networks in Azure to reach Azure services directly over the Azure backbone network**. It extends the identity of the virtual network to the service, allowing traffic to flow from the virtual network to the Azure service over the Microsoft backbone network, instead of going over the internet.
   * **Use Case:** Service Endpoints are used to secure Azure services from the public internet and to enhance security for communication between virtual networks and Azure services.
   * **Example:** You might use a Service Endpoint to connect a virtual network to Azure Storage, allowing resources in the virtual network to access Azure Storage without traffic leaving the Azure backbone.
2. **Private Endpoint:**
   * **Definition:** A Private Endpoint is a network interface that connects to an Azure service over a private connection.
   * It uses a private IP address from your virtual network, effectively bringing the service into your virtual network.
   * **Use Case:** Private Endpoints are used to access Azure services like Azure SQL Database, Azure Storage, and Azure Cosmos DB over a private and secure connection rather than over the internet. This enhances security and ensures that traffic doesn't traverse the public internet.
   * **Example:** If you have an Azure SQL Database, you can create a Private Endpoint for it. This allows applications in your virtual network to access the database using its private IP address.
3. **Private Link:**
   * **Definition:** Private Link is a service that enables you to access Azure PaaS services (like Azure Storage, Azure SQL Database, etc.) and your own services over a private connection. It works by creating a Private Link Service and a Private Link Connection.
   * **Use Case:** Private Link allows you to access services privately from your virtual network. It also allows you to expose your own services to other customers securely.
   * **Example:** If you have an Azure Storage account, you can create a Private Link Service for it. This gives you a Private Link Connection that you can use to connect to the storage account privately from your virtual network.

Scope:

Service Endpoint: Connects a virtual network to an Azure service.

Private Endpoint: Connects a virtual network to a specific instance of an Azure service.

Private Link: Provides a secure way to connect to Azure PaaS(Azure storage, Azure DB) services and your own services.

Access Method:

Service Endpoint: Allows access over the Azure backbone network.

Private Endpoint: Provides access over a private connection using a private IP address.

Private Link: Enables access to services and services exposing functionality over a private IP address.

How to access private endpoint to publish public access?

1. **Create a Private Endpoint:**
   * First, you need to create a Private Endpoint for the specific service you want to access. This involves specifying the target resource (e.g., Azure SQL Database, Azure Storage Account) and configuring the Private Endpoint settings.
2. **Configure DNS Resolution:**
   * When you create a Private Endpoint, it is associated with a private IP address. You need to configure DNS resolution to ensure that the DNS name of the service resolves to the private IP address of the Private Endpoint.
3. **Network Routing:**
   * Ensure that the necessary network routing is in place within your virtual network. Traffic to the service's DNS name should be directed to the Private Endpoint's private IP address.
4. **Access from Resources in the Same Virtual Network:**
   * Resources within the same virtual network as the Private Endpoint can access the service using its DNS name or IP address. They don't need public IP addresses, and the access is secured over the Azure backbone network.
5. **Authentication and Authorization:**
   * Private Endpoints enhance security, but you still need to handle authentication and authorization. Depending on the service, this might involve using credentials, managed identities, or other authentication methods.

It's important to note that accessing a service via Private Endpoint does not involve exposing the service to the public internet. If you want to allow access from outside your virtual network, you might consider one of the following options:

* **VPN or ExpressRoute:** If you have on-premises resources or resources in another network, you can use VPN or ExpressRoute to establish a connection to your Azure virtual network.
* **Azure Bastion:** If you need secure access to virtual machines within the virtual network, consider using Azure Bastion to connect to those VMs.
* **Hybrid Connection:** For some services, you might be able to use a Hybrid Connection to establish secure communication between on-premises resources and the service.